國 立 虎 尾 科 技 大 學
機 械 設 計 工 程 系

專 題 製 作 報 告

強化學習在機電系統中之應用

Application of reinforcement learning in mechatronic systems

指導教授： 嚴 家 銘 老 師
班 級： 四 設 三 甲
學 生： 李 正 揚 (40723110)
林 于 哲 (40723115)
黃 奕 慶 (40723138)
鄭 博 鴻 (40723148)
簡 國 龍 (40723150)

中華民國 １１０ 年 ３ 月

摘要

產業中需要加速許多工法的演算，以達到最佳化，但不能以實體一直測試不同方法，成本與時間不允許，便可以利用許多感測器觀測數值，以類神經網路運算，在虛擬環境架設結構，遠端控制、更改數值。

此專題是利用現成裝置冰球台，設置對應虛擬模擬環境，減少現實模擬參數設置、成本，再加入類神經網路之中的 Policy gradient 與 Reinforcement Learning，訓練冰球達到對應最佳化。

關鍵字:Policy gradient、虛擬環境架設結構、Reinforcement Learning

誌謝

致謝內容

目　錄

圖 表 目 錄

第一章 前言

1.1 研究動機

機器學習與機電系統結合的運用越來廣泛，我們將其運用在桌球機上，使完成訓練的機電系統能獨自與玩家對打。

1.2 研究目的與方法

本研究分兩大部分，第一運用 Open AI Gym 的 Pong-v0 來訓練機器學習，第二利用 CoppilaSim 模擬環境並訓練出能與人對打的機電系統。

利用 Gym 的訓練環境來側試不同的算法所得到的訓練結果，比較不同算法、參數間的差異，並找出較適合 Pong game 的算法、參數，循序漸進提高環境的真實程度，來減少一開始就是以實體的方式測試所帶來硬體、程式、時間和金錢等成本。

將 Gym 的訓練環境轉換到 CoppilaSim 模擬環境，利用貼近真實的模擬環境來修正在純程式的架構與真實環境間的誤差，雖然 CoppilaSim 模擬環境與真實環境仍有些微的落差，但與 Gym 相比 CoppilaSim 的環境已非常貼近真實了，拉近了虛擬與現實間的距離，提高了實用性的價值。

1.3 未來展望

1.4 規則說明

Pong game 的遊戲規則簡單，透過擊錘將球打入對方球門即得一分，只要其中一方得 21 分就結束該局。擊錘只能沿單方向來回移動來進行防守和進攻。

遊戲規則如下：

1. 球打入敵方即得一分。
2. 擊錘只單一方向移動。
3. 最快贏得 21 分者獲勝，並結束該局遊戲。

第二章 機器學習的訓練方法

2.1 類神經網絡

類神經網路的概念來源是觀察人類的中樞神經系統而啟發的，類神經網路主要由多個節點組成，每個節點稱為神經元（neurons），神經元與神經元連結而形成的網路稱為神經網路。神經元主要又分成三層：輸入層（input layer）、隱藏層（hidden layer）和輸出層（output layer）。神經元會接收上一層的訊號，並經過啟動函數（activation function）計算並輸出至下一層神經元。神經網路輸入層的神經元負責接收訊號，透過隱藏層來運算，再傳遞到輸出層執行動作。由於每個神經元經過啟動函數的非線性計算，所以神經元傳出的數值也是非線性。神經元與神經元間連線在生物學稱為突觸（Synapse），在數學模型裡每個突觸都有權重（weights）。在既定框架下，調配權重值和啟動函數會使結果更接近預期。神經網路的架構指標為以下幾項：階層數、每層神經元個數、神經元連接方式、啟動函數的類型等。

神經訊號傳遞分為前饋（feed-forward）和回饋（backpropagation）。前饋為傳遞經過啟動函數計算的數值。回饋則是修正權重和偏差（biases），以提高下次前饋計算的準確度。

2.1.1 啟動函數

以下介紹幾種較為常見的啟動函數及其特性：

- Sigmoid Function：
  輸出介於 0 到 1 之間，適用於二元分類，方程式具有非線性、可連續微分、並具有固定輸出範圍等特性，並可以讓類神經網路呈現非線性。

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- SigmoidPrime Function：
  這是從 Sigmoid Function 微分得來，以梯度運算的方式，可以減少梯度誤差，但也是造成梯度消失的主要原因，若要改善梯度消失需要搭配優化器使用，方程式如下:

$$\sigma^{'}(x) = \sigma(x)[1 - \sigma(x)]$$

- Softmax：
  Softmax 會計算每個事件分布的機率，適用多項目分類、其機率總合為 1。以此專案為例，假設擊錘移動有向上移動、向下移動及不移動這三個決策選項，則這三個決策機率值總和為 1。

$$S(x) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_i}}$$

- ReLU Function：
  ReLU Function 方程式特性：若輸入值為負值，輸出值為 0；若輸入值為正值，輸出則維持該輸入數值。ReLU 計算方式簡單、收斂速度快，這是類神經網路最普遍拿來使用的啟動函數，因為可以解決梯度消散的問題，但須注意：起始值若設定到不易被激活範圍或是權重過度所導致權重梯度為 0 就會造成神經元難以被激活。

$$f(x) = max(0, x)$$
$$if, x < 0, f(x) = 0$$
$$elsef(x) = x$$

- Adam Function
  結合了 Adagrad 和 RMSprop 的優勢. 有論文表示，在訓練速度方面有巨大性的提升，但在某些情況下，Adam 實際上會找到比隨機梯度下降法更差的解決方法。以下是計算過程:

$$g_t = \delta_\theta f(\theta)$$

  First moment exponentially moving averages : $m_t = \beta(m_{t-1}) + (1-\beta_1)(\nabla w_t)$

  $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$

  Second moment exponentially moving averages :$v_t = \beta_2(v_t - 1) + (1 - \beta_2)(\nabla w_t)^2$

  $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$

  Hence,Adam Function:

$$\omega_{t-1} = \omega_t - \frac{\eta}{\sqrt{\hat{v}_t - \epsilon}}\hat{m}_t$$

- Mean Squared Error
  他能告訴你一組點與回歸線接近的程度，透過獲取點與回歸線之距離 (這些距離就是誤差) 並對它們進行平方來做到這點，而平方是為了消除所有負號，也能讓更大的差異賦予更大的權重。(Extracted from here).

  回歸線: 數據點間最小距離的一條線。

  n : 數據點的數量

  y_i : 觀測值

  $\hat{y}_i$: 預測值

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y}_i)^2$$

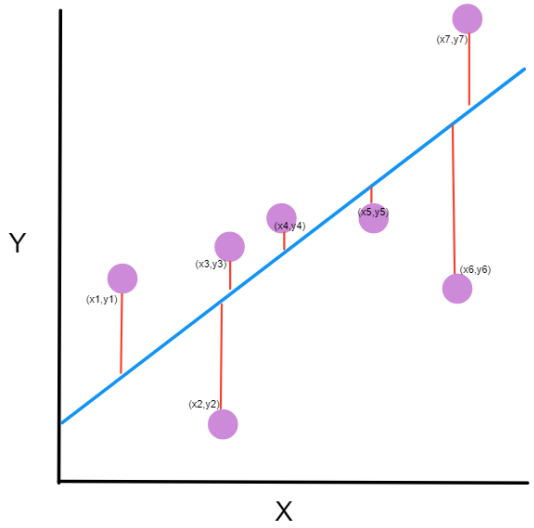


圖. 2.1: regression line

2.1.2 優化器

為了讓機器學習的錯誤率降低，因此利用優化器來降低 loss function 的值，在 error surface 上找到最小值，即是找到錯誤率最低的地方。以下將介紹幾種優化的方法：

- Gradient Descent

  利用梯度的方式尋找最小值的位置，其特色可找到凸面 error surface 的絕對最小值，在非凸面 error surface 上找到相對最小值。其缺點是在非凸面 error surface 要避免被困在次優的局部最小值。

- Batch gradient descent

  用批次的方式計算訓練資料，整個資料集計算梯度只更新一次，因此計算和更新時會占用大量記憶體。整體效率較差、速度較緩慢。由 Gradient Descent 延伸出來的算法。其收斂行為與 Gradient Descent 相同。

- Stochastic gradient descent

  每次執行時會更新並消除誤差，有頻繁更新和變化大的特性，較不容易困在特定區域。由 Gradient Descent 延伸出來的算法。其收斂行為與 Gradient Descent 相同。

- Mini-batch gradient descent

  結合 Batch gradient descent 和 Stochastic gradient descent 的特點：批量計算和頻繁更新，所衍伸的算法。利用小批量的方式頻繁更新，並使收斂更穩定。其缺點：學習率挑選不易、預定義 threshold 無法適應數據集的特徵、對很少發生的特徵無法執行較大的更新、非凸面 error surface 要避免被困在次優的局部最小值等。

- Gradient descent optimization algorithms

  為了改善前面幾種算法而發展出來的優化算法。以下將列出數種優化算法。

- Momentum

  在梯度下降法加上動量的概念，會加速收斂到最小值並減少震盪。

- Nesterov accelerated gradient

  NAG，有感知能力的 Momentum：在坡度變陡時減速，避免衝過最小值所造成的震盪 (為了修正到最小值，來回修正而產生的震盪)

- Adagrad

  其學習率能適應參數：頻繁出現的特徵用較低的學習率，不經常出現的特徵則用較高的學習率，且無須手動調整學習率。其缺點是，學習率會急遽下降，最後會無限小，這算法就不再獲得知識。

- Adadelta

  為 Adagrad 的延伸，下降激進程度，學習率從更新規則中淘汰，不需設定預設學習率。

- RMSprop

  為了解決 Adagrad 學習率急劇下降的問題，學習率除以梯度平方的 RMS，解決學習率無限小的情形。

- Adam

  類似 Momentum，更加穩定快速的收斂。

- AdaMax

  與 Adam 相似，依靠 (u_t) 最大運算

- Nadam

  結合 Adam 和 NAG ，應用先前參數執行兩次更新，一次更新參數一次更新梯度。

- AMSGrad

  改善 Adam 算法所導致收斂較差的情況 (用指數平均會減少其影響)，換用梯度平方最大值來做計算，並移除去偏差的步驟。是否有比 Adam 算法好仍有待觀察。

- Gradient noise

  有助於訓練特別深且復雜的網絡，noise 可改善不良初始化的網路。

2.2 深度學習

- Deep learning 會在此時崛起?
  電腦科學對於深度學習電的依賴越來越重要的四個主要趨勢。

  (1) 支援張量運算程式語言與程式庫的崛起
  首先，自從 1960 年代開始，諸如 APL、MATLAB、R 和 Julia 等領域的特定語言的發展轉變為多為數組 (通常稱為張量) 進入一組由全為數學語 (或運算符) 支援的第一類物件以對其進行操作
  此外，基於數組的編程能夠以通用語言 (如 Python、Lisp、C++ 和 Lua) 高校運作，諸如 NumPy、Torch、Eigen 和 Lush 等

  (2) 自動求導數套件的開發
  其次，自動微分的發展使得完全自動化計算導數的艱鉅工作成為了可能，這使的運用不同的機器學習方法進行實驗變得非常容易，同時仍然允許基於梯度的優化。因為 autograd 軟件包的普及，並將該技術用於 NumPy 數組，並且類似的方法也用於框架中，例如: Chainer, DyNet, Lush, Torch, Jax and Flux.jl.

(3) 自由開源軟體的普及

第三隨著自由軟件的到來，科學界從封閉的專用軟件 (例如 Matlab) 轉向開源，Python 的生態系統，帶有 NumPy、SciPy 和 Pandas 等軟件包，滿足了大多數研究人員對數值分析需求，同時讓他們能夠利用龐大的程式庫來處理數據集的預處理，例如: 統計分析、繪圖等。

此外，自由軟體的開放性、互相操作性和靈活性，促進了充滿活力的社會發展，這樣社會可以通過擴展程式庫現有的功能，或著在需要時通過開發和發布全新的程式庫功能，來迅速滿足新的或不斷變化的需求。儘管有豐富的非 Python 語言支持的神經網路開源軟件，從 Lish 中的 Lush、C++ 中的 Torch、Objective-C 和 Lua、C++ 中的 EBLearn、C++ 中的 Caffe 開始，大型生態系統 (諸如 Python) 的網路效應，促使研究快速發展成為一項必不可少的技能。因此，自 2014 年以來，大多數深度學習的框架都將 Python 介面作為一項基本功能進行融合。

(4) 多核 GPU 運算的發展

最後，利用大規模通用硬體 (例如 GPU) 的可能性，提供深度學習方法所需的計算能力、專用的程式庫 (例如:cuDNN) 以及大量學術著作 (例如 Andrew Lavin、maxdnn: 使用 maxwell gpus 進行深度學習的高效卷積內核、2015 年 1 月以及 Andrew Lavin 和 Scott Gray 用於卷積神經網路的快速算法、2016 年 IEEE 電腦視覺和圖形識別 (CVPR)，第 4013-4021 頁，2016), 產生了一系列高性能可重用的深度學習內核，這些內核使諸如 Caffe,Torch7, or TensorFlow 的框架能夠利用這些硬體加速器。

2.3 強化學習

強化學習是通過 agent 與已知或未知的環境持互動，不斷適應與學習，得到的回饋可能是正面，也就是 reward，如果得到負面，那就是 punishments。考慮到 agent 與環境互動，我們就能決定要執行哪個動作。簡而言之，強化學習是建立在 reward 與 punishments 上。

與其他機器學習不同，因為我們不需要在意訓練數據集。互動不是發生在數據上，而是在環境下發生，通過互動描繪出真實的場景。由於強化學習是建立在環境的，因此許多參數進行運算，需要大量信息來學習，並根據此採取行動。強化學習的環境是真實世界，可能是 2D 或 3D 模擬世界的場景，也可能是建立於遊戲的場景。從某種意義上來說，強化學習的範圍很廣，因為環境的規模可能很大，並且在環境中有多相關因素，影響著彼此。強化學習的目的是達到目標。強化學習的獎勵是來自環境。

- 強化學習涵蓋範圍

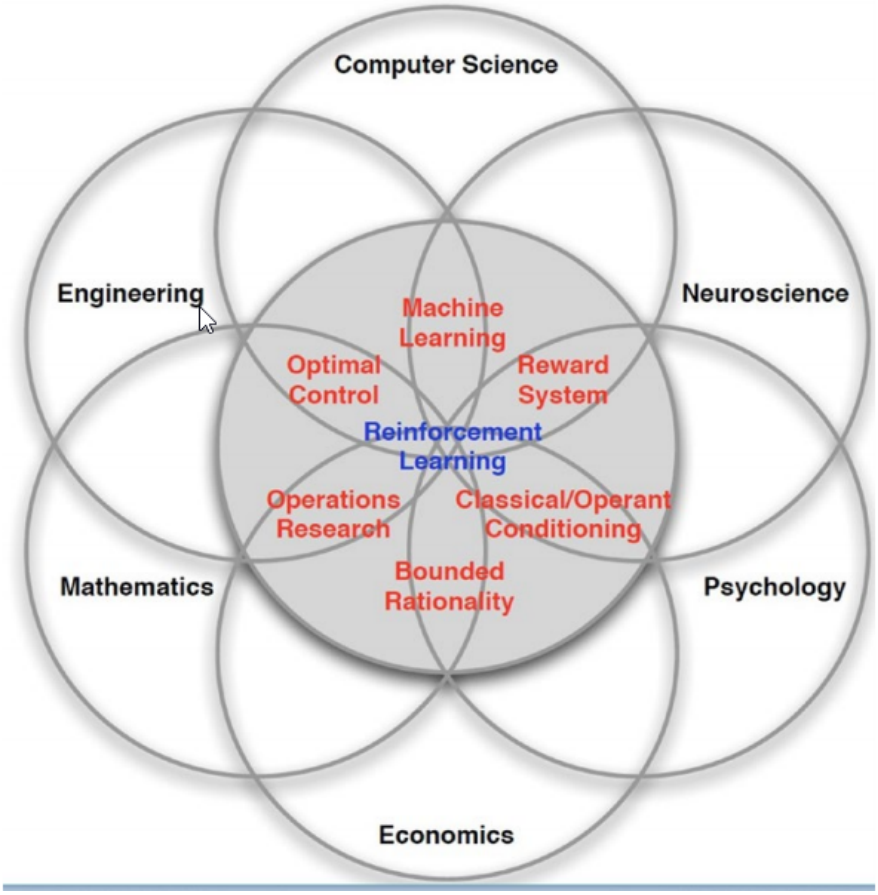


圖. 2.2: Venn diagram;

- 強化學習的流程
  需要考慮的重點:
  - 強化學習的週期是互相聯繫的
  - 分明的溝通是在考慮獎勵的情況下發生的
  - agent 與環境之間存在著獨特的溝通。

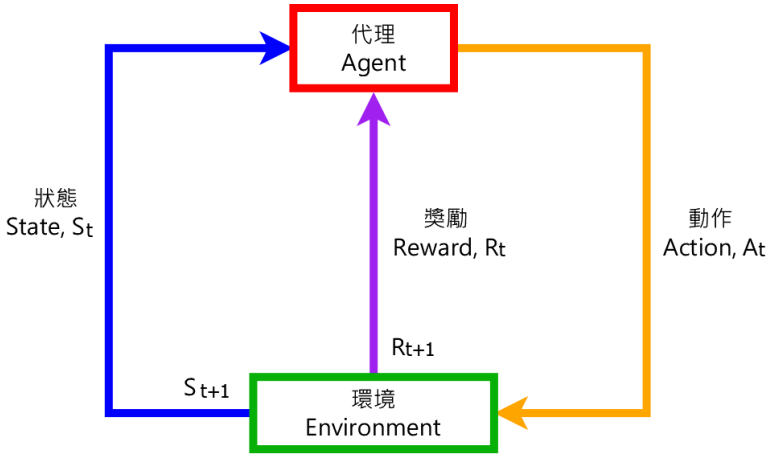


圖. 2.3: RL structur

– 目標或機器人從一狀態移動到另一狀態。

– 從一狀態轉移到另一種狀態採取動作。

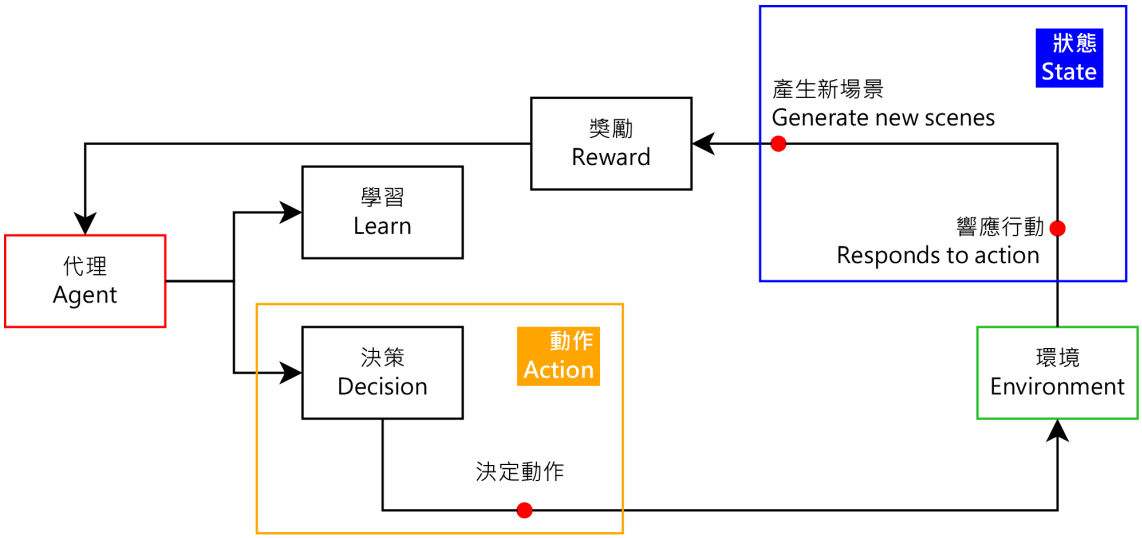


圖. 2.4: The entire interaction process;

agent 是決策者，因為他會試圖採取獲得最高大獎勵的行動。
當 agent 開始與環境互動時，他可以選擇一個操作並做出相應的回應，從這時起，將會創建新的場景，當 agent 從一個環境變為另一個環境中，每項更改都會導致某種修改，這些變化被描述為場景，每個步驟中發生的過度都有助於 agent 更有效地解決強化學習的問題。

- 強化學習中各種術語
在此情況下，有兩個很重要的常數-γ、λ

γ 用於每個狀態轉換，且在每次狀態變化時，都是一個恆定值，γ 會從你將在每個狀態獲得的獎勵類型中，告訴你訊息。γ 又叫衰減因子，他決定了我們未來可獲得的獎勵類型:

– γ 值為 0 表示獎勵僅與當前狀態有關係。

– γ 值為 1 時，代表獎勵是長期的。

當我們處理時間問題時，通常使用 λ，這是涉及更多的連續狀態的預測，每種狀態下的 λ 值增加，代表我們的算法學習速度很快，適用強化學習技術時，更快的算法會產生更好的結果。

- 與強化學習的互動
  agent 和環境之間的互動會產生獎勵，我們採取行動，從一種狀態轉移到另一種狀態
  強化學習是一種實現如何將情況映設為行動的方法，從而最大化並找到獲得最高獎勵的方法，機器或機器人不會像其他機器學習形式的機器人那樣被告知要採取哪些行動，但機器必須發現哪些動作會產生最大的獎勵。
- 獎勵如何運作
  獎勵是我們從一狀態過渡到另一狀態時所獲得的動力，它可以是分數，例如在遊戲中。我們訓練的次數越多，我們就變得越精準，且我們的回報越大。
- Agents
  在強化學習方面，agent 是一個軟體程式，能做出明智的選擇，應能購感知到正在環境發生的事情，以下是 agent 基本步驟:
  – 當 agent 可以感知環境時，它可以做出比以往更好的決定。
  – agent 採取的決定就會產生行動。
  – agent 執行的行動必須是最好最佳的。

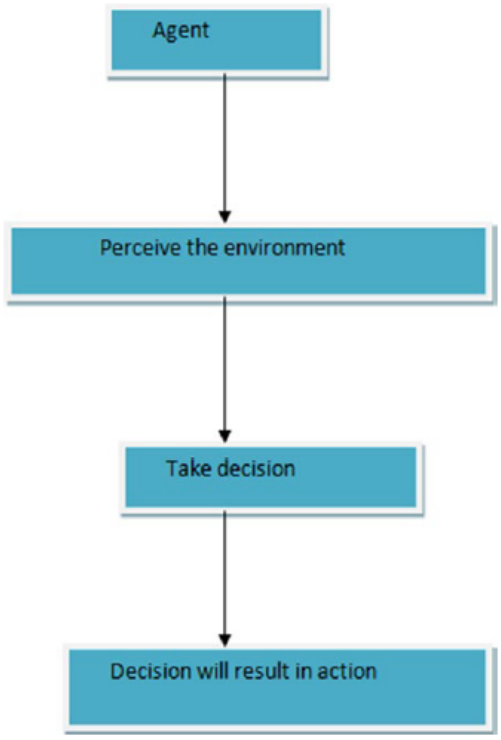


圖. 2.5: agent

- 強化學習的環境
  強化學習中的環境由某些因素組成，會對 agent 產生影響，agent 必須根據環境適應其各種因素，並做出最佳決策，這些環境可以是 2D 世界或是網格，甚至是 3D 世界。
  以下是環境的一些重要功能:
  - 具有確定性。
  - 可觀察性。
  - 是離散或是連續的狀態。
  - 單個 agent 或是多個 agent。

2.4 深度強化學習

2.4.1 Deep Reinforcement Learning

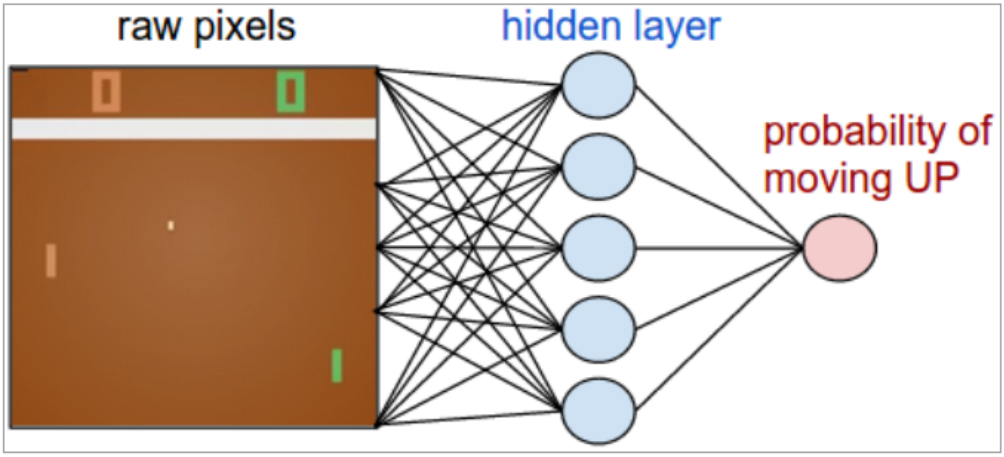


圖. 2.6: Our network is a 2-layer fully-connected net.

我們將定義一個可以執行我們的玩家 (agent) 的類神經網路，該網路將獲取遊戲狀態並決定我們應該做什麼 (向上移動或向下移動) 我們使用一個 2 層神經網路，該網路獲取原始圖像像素 (100,800 個數字 (210*160*3))，並生成一個表示上升概率的數字。使用隨機策略是標準做法，這意味著我們只會產生向上移動的可能性，每次迭代時，我們都會從該分布中採樣 (即扔一枚有偏見的硬幣) 已獲得實際移動。

2.4.2 Supervised Learning

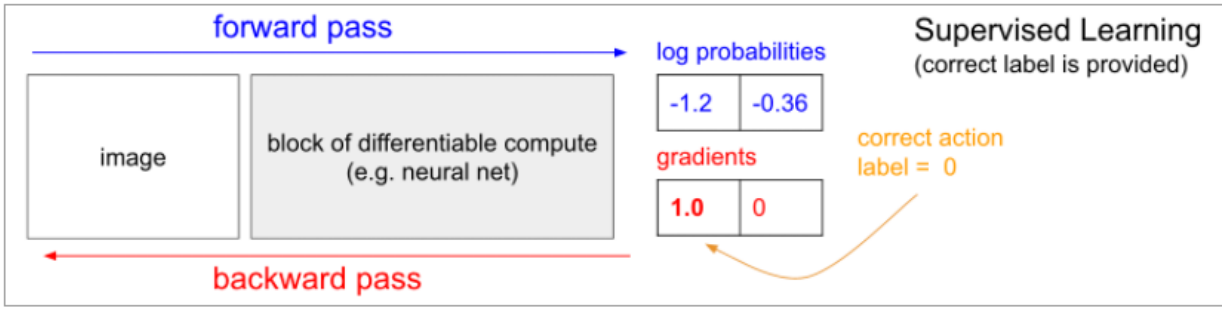


圖. 2.7: supervising learning

在我們滲入探討 Score Function 解決方案之前，需要簡短介紹有關監督學習的知識，因為正如我們看到的，與我們架構類似，在普通通的監督學習中，我們會將圖像送到網路，並獲得伊些概率，例如對於兩個類別的上和下。這裡顯示的是向上和向下對數概率 (-1.2,-0.36)，而不是原始機率 (在這個情況下，是 30% 和 70%)，因為我們總是優化正確標籤的對數概率 (這使我們的演算法更好，病等效於優化原始概率，因為對數是單調的)，而在監督學習中，我們將可以獲取標籤，例如:

我們可能被告知現在正確的做法是向上運動 (標籤 0)，在執行過程中，我們將以上的對數機率輸入 1.0 的梯度，然後運行 backprop 來計算梯度向量

$Wlogp(y = UP|x)$ 這個梯度將告訴我們應如何更改百萬個參數中的每個參數，使網路預測往上的可能性更高，例如: 網路中的百萬個參數之一可能具有-2.1 的梯度，這意味著如果我們將該參數增加一個小的正值 (例如 0.001)，則往上的對數機率將因 2.1*0.001 而降低 (由於負號而減少)，如果我們隨後更新了參數，當之後遇到非常相似的圖像時 (也就是環境狀況)，我們的網路現在更有可能預測往上。

2.4.3 對數導數技巧

機器學習涉及操縱機率。這個機率通常包含 normalised-probabilities 或 log-probabilities。能加強解決現代機器學習問題的關鍵點，是能夠巧妙的在這兩種型式間交替使用，而對數導數技巧就能夠幫助我們做到這點，也就是運用對數導數的性質。

2.4.4 Score Functions

對數導數技巧的應用規則是基於參數 θ 梯度的對數函數 $p(x:\theta)$，如下:

$$\nabla_\theta logp(x:\theta) = \frac{\nabla_\theta p(x:\theta)}{p(x:\theta)}$$

$p(x:\theta)$ 是 likelihood ; function 參數 θ 的函數，它提供隨機變量 x 的概率。在此特例中，$\nabla_\theta logp(x:\theta)$ 被稱為 Score Function，而上述方程式右邊為 score ratio(得分比)。

score function 具有許多有用的屬性:

- 最大概似估計的中央計算。最大概似是機器學習中使用的學習原理之一，用於廣義線性回歸、深度學習、kernel machines、降維和張量分解等，而 score 出現在這些所有問題中。
- score 的期望值為零。對數導數技巧的第一個用途就是證明這一點。

$$\mathbb{E}_{p(x;\theta)}[\nabla_\theta \log p(\mathbf{x};\theta)] = \mathbb{E}_{p(x;\theta)}\left[\frac{\nabla_\theta p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)}\right]$$

$$= \int p(\mathbf{x};\theta)\frac{\nabla_\theta p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)}dx = \nabla_\theta \int p(\mathbf{x};\theta)dx = \nabla_\theta 1 = 0$$

  在第一行中，我們應用了對數導數技巧，在第二行中，我們交換了差異化和積分的順序，這種特性是我們尋求概率靈活性的類型: 它允許我們從期望值為零的分數中減去任何一項，且此修改不會影響預期得分 (控制變量)。

- 得分的方差是 Fisher 信息，用於確定 Cramer-Rao 下限。

$$\mathbb{V}[\nabla_\theta \log p(\mathbf{x};\theta)] = \mathcal{I}(\theta) = \mathbb{E}_{p(x;\theta)}[\nabla_\theta \log p(\mathbf{x};\theta)\nabla_\theta \log p(\mathbf{x};\theta)^\top]$$

  我們現在可以從對數概率的梯度躍升為概率的梯度，然後返回，但是真正要解決的其實是計算困難的期望梯度，所以我們可以利用新發現的功能:score function 為此問題開發另一個聰明的估計器。

2.4.5 Score Function Estimators

我們的問題是計算函數 f 的期望值的梯度：

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \nabla_\theta \int p(z;\theta)f(z)dz$$

這是機器學習中的一項常態性任務，在變數推理中進行後驗計算，在強化學習中進行價值函數和策略學習，在計算金融中進行衍生產品定價以及在運籌學中進行庫存控制等。該梯度很難計算，因為積分通常是未知的，我們計算梯度所依據的參數 θ 的分佈為 p(z;θ)，此外，當函數 f 不可微時，我們可能想計算該梯度，使用對數導數技巧和得分函數的屬性，我們可以更方便地計算此梯度：

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \mathbb{E}_{p(z;\theta)}[f(z)\nabla_\theta \log p(z;\theta)]$$

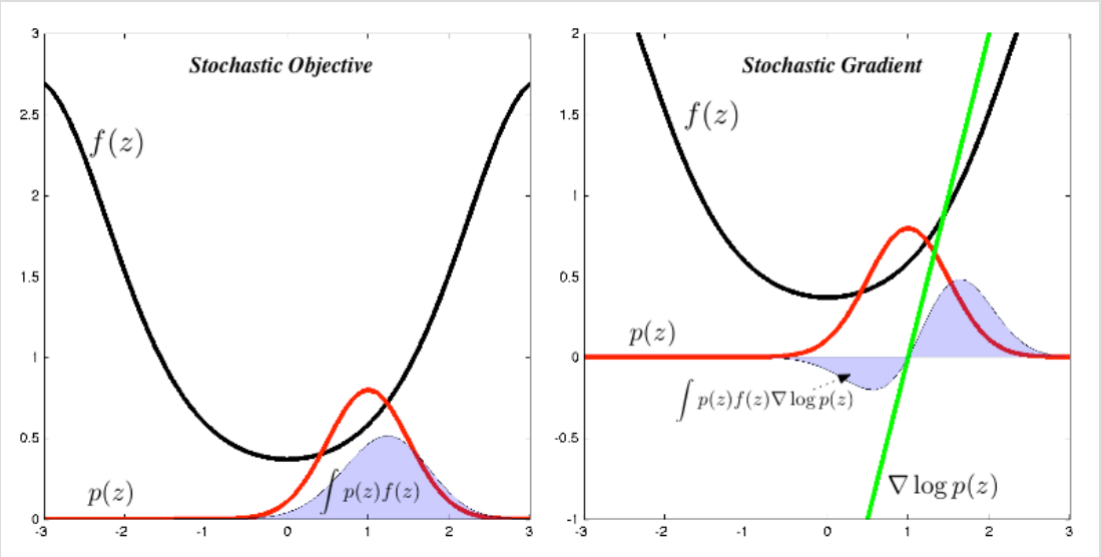


圖. 2.8: gradient change

讓我們導出該表達式，並探討它對我們的優化問題的影響。為此，我們將使用另一種普遍存在的技巧，一種概率恆等的技巧，在該技巧中，我們將表達式乘以 1，該表達式由概率密度除以自身而形成。將特性技巧與對數導數技巧相結合，我們獲得了梯度的得分函數估計量:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \int \nabla_\theta p(z;\theta) f(z) dz$$

$$= \int \frac{p(z;\theta)}{p(z;\theta)} \nabla_\theta p(z;\theta) f(z) dz$$

$$= \int p(z;\theta) \nabla_\theta \log p(z;\theta) f(z) dz = \mathbb{E}_{p(z;\theta)}[f(z)\nabla_\theta \log p(z;\theta)]$$

$$= \int p(z;\theta) \nabla_\theta \log p(z;\theta) f(z) dz = \mathbb{E}_{p(z;\theta)}[f(z)\nabla_\theta \log p(z;\theta)]$$

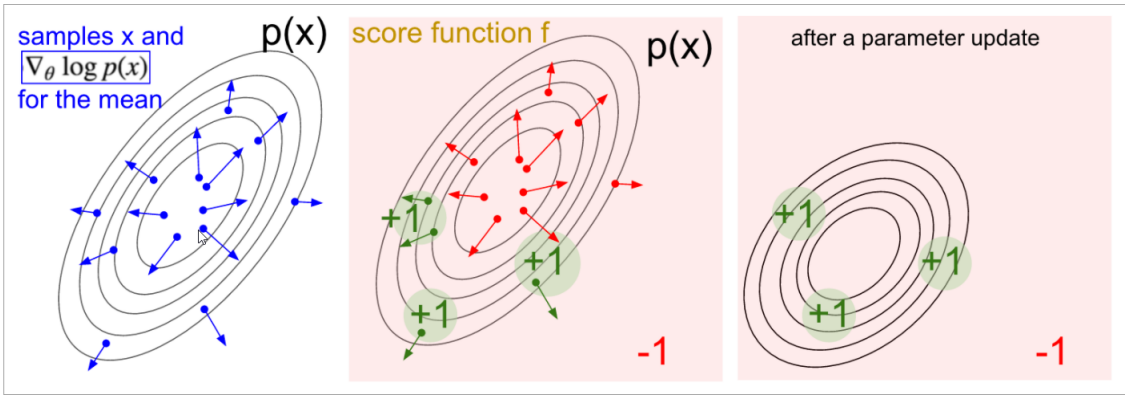
$$\approx \frac{1}{S} \sum_{s=1}^{S} f(z^{(s)}) \nabla_\theta \log p(z^{(s)};\theta) \quad z^{(s)} \sim p(z)$$

在這四行中發生了很多事情。在第一行中，我們交換了導數和積分。在第二

行中，我們應用了概率身份技巧，這使我們能夠形成得分比，然後使用對數導數技巧，用第三行中對數概率的梯度替換該比率。這在第四行給出了我們所需的隨機估計量，這是由蒙特卡洛計算的，方法是首先從 p（z）提取樣本，然後計算加權梯度項。.

更簡單的描述，我們有一些分佈 $p(x;\theta)$（我使用了速記 $p\ x$ 來減少混亂），我們可以從中採樣（例如，這可能是高斯）。對於每個樣本，我們還可以評估分數函數 f，該函數將樣本作為樣本並給出標量值。該方程式告訴我們，如果我們希望其樣本達到較高的分數（由 f 判斷），應該如何改變分佈（通過其參數 θ)，特別是，它看起來像：畫出一些樣本 x，評估其分數 f（x），並且對於每個 x 也評估第二項 $\nabla_\theta logp(x;)$，那第二項是什麼，它是一個向量-漸變為我們提供了參數空間中的方向，這將導致分配給 x 的概率增加。換句話說，如果我們要在的方向上微移 θ，$\nabla_\theta logp(x;)$，我們會看到分配給 x 的新概率略有增加。如果回顧一下公式，它告訴我們應該朝這個方向發展，並將標量值加到上面 $f(x)$。這樣一來，得分較高的樣本將比那些得分較低的樣本“拖拉”更強的概率密度，因此，如果我們要根據 p 上的幾個樣本進行更新，則概率密度將朝著較高分數的方向移動，從而使得分較高的樣本更有可能出現。

score function gradient estimator 的可視化，左：高斯分佈及其中的一些



樣本（藍點)，在每個藍點上，我們還繪製了相對於高斯平均參數的對數概率的梯度，箭頭指示應微調分佈平均值以增加該樣本概率的方向。中間：某些得分函數的疊加，在某些小區域中除了 +1 之外，其他所有地方都給出-1（請注意，這可以是任意的，不一定是可微分的標量值函數)，箭頭現在採用了顏色區別，因為由於更新中的乘法運算，我們將平均所有綠色箭頭和紅色箭頭的負數。右：更新參數後，綠色箭頭和反向紅色箭頭將我們向左移至底部。現在，根據需要，該分佈中的樣本將具有更高的預期分數。

2.5 比較個方法的特色

第三章 馬可夫決策

3.1 Markov Chain

當前決策只會影響下個狀態，當前狀態轉移 (action) 到其他狀態的機率有所差異。

3.2 Markov Reward Process

action 到指定狀態會獲得獎勵。

$$R(s_t = s) = \mathbb{E}[r_t | s_t = s]$$

$$\gamma \in [0, 1]$$

- Horizon：
  在無限的狀態以有限的狀態表示
- Return：
  越早做出正確決策獎勵越高
  $$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... + \gamma^{T-t-1} R_T$$
- State value function：
  決策價值
  $$V_t(S) = \mathbb{E}[G_t | s_t = s]$$

$$P(s_{s+1} = s' | s_t = s, a_t = a)$$

3.3 Markov Decision Process

在 MRP 中加入決策 (decision) 和動作 (action)

- S：state 狀態
- A：action 動作
- P：狀態轉換

$$P(s_{s+1} = s' | s_t = s, a_t = a)$$

第四章 訓練環境

4.1 openAI Gym

Gym 是用於開發和比較強化學習算法的工具包，他不對 agent 的結構做任何假設，並且與任何數據計算庫兼容，而可以用來制定強化學習的算法。這個環境具有共享的介面，使我們能用來編寫常規算法，也就能教導 agents 如何步行到玩遊戲。

4.2 Pong

取自 1977 年發行的一款家用遊戲機 ATARI 2600 中的遊戲，內建於 Gym，這是一個橫向的乒乓遊戲，左方是遊玩者，右邊是馬可夫決策的特例，每個邊緣都會給予 reward(figue1)，目標就是計算再任意階段動作最佳路徑，已獲得 rewardd 最大值。

4.3 為什麼選擇 score function 算法

多篇論文已經廣泛使用了 ATARI 遊戲並結合了 DQN(它是一種在強化學習算法裡，知名度較高的)，事實證明，Q-Learning 並不是一個很好的算法，實際上大多數人比較喜歡使用 Policy Gradients，包括原始 DQN 論文的作者，他們在調優後顯示 Policy Gradients 比 Q 首選 PG，因為它是端到端的：有一個明確的政策和一種有原則的方法可以直接優化預期的回報。但是礙於時間考量，而選擇了類似 PG 的算法，也就是 score function gradient estimator(取用 Andrej Karpathy)，從像素開始，通過深度神經網絡使用 ATARI 遊戲（Pong），並在整個過程結合 numpy，作為訓練工具。

4.4 Pong from pixels

左：乒乓球遊戲。右：Pong 是 Markov 決策過程（MDP）的特例：圖，其中每個節點都是特定的遊戲狀態，每個邊緣都是可能的（通常是概率性的) 改變，每條邊界都給與獎勵，目標是計算在任何狀態下發揮作用的最佳方式，以最大限度地提高獎勵。
Pong 的遊戲是簡單的強化學習中，很好的例子，在 ATARI 2600 版本中，我們會控制右邊邊的操縱板 (左邊由不錯的 AI 控制)。在較低級別上，遊戲的運行方式如下：我們收到一個圖像幀（一個 210x160x3 byte 數組（從 0 到 255 的整數給出像素值)），然後決定是否要向上或向下移動操縱板（即二進制選擇）。每次選擇之後，遊戲模擬器就會執行動作並給予我們獎勵: 如果球超過了對手，則為 +1 獎勵；如果我們錯過球，則為-1 獎勵；否則為 0。當然，我們的目標是移動球拍，以便獲得很多獎勵。

第五章　模擬環境

第六章 伺服器

- Oracle VM VirtualBox
  因應在虛實整合需要用到虛擬環境去模擬的情況，我們使用 Ubuntu 當作我們的作業系統去建構虛擬環境，而在此之前要先安裝「虛擬機器工作站」—Oracle VM VirtualBox 。

6.1 Oracle VM VirtualBox 介紹

當要使用不同的的作業系統 (Linux , Ubuntu , Red Hat ...) 而不想與其資料存放時共用一個硬碟 (沒有多餘硬碟, 不想硬碟之間有資料重疊...) 時，就可以使用其軟體做練習，降低操作失誤帶來的成本，而此軟體目前為免費，並隨時會更新，另外其特色有:

- 只要自備作業系統 (光碟片, ISO 映像檔) ，即可在啟動 Oracle VM VirtualBox 後直接開啟要操作的執行檔 (作業系統)，不必再把主機本身重新關機，當然開啟多個作業系統之間也有共通性，可直接從視窗 A 做網路、檔案分享、複製貼上等動作到視窗 B。
- 除了作業系統裡面的執行，還可在其中練習磁碟分割、格式化以及 BIOS 啟動等 (但是未支援 USB 啟動) 。
- 空間的佔用上並不是真實佔用空間，而是依據使用者的操作而變化 (使用者用多少就是多少)。相對的，使用者雖然一開始設定該虛擬電腦的記憶體大小與硬碟空間是實時依據操作者決定，但終究還是佔掉電腦效能，所以 VirtualBox 的效能還是依據電腦本身的硬體配備
-

第七章 深度強化學習的訓練與控制

程式訓練流程：

擷取影像，將影像裁剪至實際遊戲範圍，並簡化像素以利提高訓練時的計算速度，減少運算時的負擔，過濾顏色只保留球與擊錘，並把取到的影像二元化，取兩幀畫面進行比較，掌握球與擊錘間的相對位置 (畫面差)，透過前饋：計算球在環境的狀態及擊錘移動的決策，畫面差透過 W1 權重來計算球在環境的狀態，透過 W2 權重並經過啟動函數 (activation function) 得出擊錘移動的決策。透過產生隨機值的方式來與擊錘移動決策值進行比較，隨機值若落在決策向上移動的區間，擊錘就會向上移動；落在決策向下移動的區間，就會進行該動作。計算 discount reward 及獎勵的加總。

在單局結束時，紀錄下該局累積下來的經驗，亦是紀錄該局所修正出來的參數而進行獎勵計算、log probability、

第八章　問題與討論

參考文獻

[1] https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e
[2] https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

作者簡介

test